Relating Clusterization Measures and Software Quality

Béla Csaba*, Lajos Schrettner†, Árpád Beszédes†, Judit Jász†, Péter Hegedűs† and Tibor Gyimóthy†
*Department of Set Theory and Mathematical Logic
University of Szeged, Hungary
Email: bcsaba@math.u-szeged.hu
†Department of Software Engineering
University of Szeged, Hungary
Email: {schrettner,beszedes,jasy,hpeter,gyimothy}@inf.u-szeged.hu

***Abstract*—Empirical studies have shown that dependence clusters are both prevalent in source code and detrimental to many activities related to software, including maintenance, testing and comprehension. Based on such observations, it would be worthwhile to try to give a more precise characterization of the connection between dependence clusters and software quality. Such attempts are hindered by a number of difficulties: there are problems in assessing the quality of software, measuring the degree of clusterization of software and finding the means to exhibit the connection (or lack of it) between the two.**

In this paper we present our approach to establish a connection between software quality and clusterization. Software quality models comprise of low- and high-level quality attributes, in addition we defined new clusterization metrics that give a concise characterization of the clusters contained in programs. Apart from calculating correlation coefficients, we used *mutual information* to quantify the relationship beetween clusterization and quality. Results show that a connection can be demostrated between the two, and that mutual information combined with correlation can be a better indicator to conduct deeper examinations in the area.

***Keywords*-Software quality model, Quality metrics, Dependence cluster, Clusterization metrics, Correlation, Mutual information**

I. Introduction

A dependence cluster is a set of program elements that mutually depend on each other [1]. Their existence in source code has been getting increasing attention recently because it has been demonstrated in various maintenance-related contexts that they may be detrimental to code comprehension, maintenance and evolution, impact analysis, and testing. However, it has not been investigated systematically yet whether the extent a system exhibits dependence clusters can be used to predict quality issues. Software quality evaluation methods usually rely on classifications such as those provided by the ISO/IEC 9126 standard [2], which provides a framework for quality models, but does not define the lower level components and the exact rules of calculation for the high level quality attributes.

The aim of the present research is to verify whether a connection can be established between the amount of clusters existing in a program and other measurable quality aspects. If so, then dependence cluster research should be oriented towards incorporating cluster information into quality evaluations and ways of avoiding them or reducing their sizes in programs. In state-of-the-art research, the main vehicle for revealing dependence clusters has been to use a graphical representation called Monotone Size Graph (MSG) [3] of a program. MSGs are drawn by first calculating *dependence sets* (the set of dependent elements) for each program element, then plotting the sizes of these sets in monotonically increasing order along the horizontal axis (see Figure 1). The MSG is excellent for expert analysis, but it is not concise enough for exhibiting any relationship between clusters and software quality. For this to be possible, we need some kind of metric that gives us an overall rating of the number and sizes of the clusters a given software system contains. We call this property of the system *clusterization*.

One metric used for clusterization is the "area under the MSG" (AREA) that has been found to be fairly useful in practice [3]. It can be demonstrated, however, that it is not ideal, and there might be other measures that perform better. For instance, one would intuitively say that the program shown on the left in Figure 1 exhibits heavier clusterization than the other, but the AREA metric says the opposite (the values are 38.9% and 40.8%, respectively, see Table I).

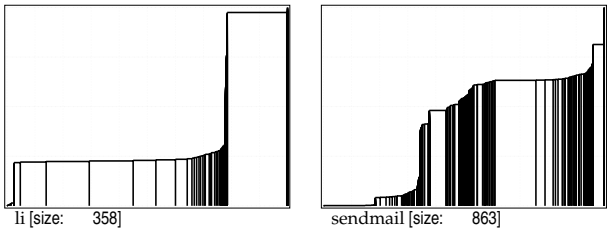

Figure 1. MSGs for programs `li` and `sendmail`, dependence sets of the same size are distinguished

In order to give a more precise characterization of the connection between clusterization and software quality, we

- Propose two alternative clusterization measures in addition to AREA: "regularity of dependence sets" (REGU) and "entropy of dependence sets" (ENTR)

- Calculate quality attributes of a set of subject systems using an existing quality model
- Relate clusterization metrics and quality attributes using correlation and mutual information analysis [4]

Although the results of the paper are still preliminary, they suggest that there is still a lot of potential in researching alternative clusterization measures and their verification against quality models.

The paper is organized as follows. In Section II we overview previous results that we build our approach on, while the actual clusterization measurement and evaluation method is presented in Section III. Section IV presents the results of the empirical evaluation, after which we conclude the paper.

II. Background

A. Related work

The phenomenon of *dependence clusters* was first described by Binkley and Harman in 2005 [3] based on program slices and Program Dependence Graphs [5]. Initially, they were defined as sets of mutually interdependent statements or, more practically, maximal sets of statements that all have the same slices. Later, set coincidence has been substituted with the approximation of set size equivalence [1]. The notion of dependence clusters can be generalized to other kinds of dependence types and different program elements at various granularity. Furthermore, it seems that dependence clusters are independent of the programming language and the type of the system [6], [7], [8]. During the investigations presented in this paper, we used SEA-based dependence clusters [7].

The current view is that large dependence clusters hinder many different software engineering activities, exhibiting a sort of "dependence anti patterns" [9]. Various software engineering areas are influenced as presented in related literature [6], [1], [10]. A recent research activity is the identification of the dependence cluster causes; specifically the identification and possible removal of the so-called *linchpin* program elements, which are seen as central in terms of dependence relations, and are often holding together the whole program. If the linchpin is ignored when following dependences, the clusters will vanish [11], [1].

Monotone Size Graphs (MSG) and the AREA metric [3] have been used to characterize dependence clusters and clusterization in programs. A related investigation was performed by Islam et al., who defined alternative descriptions of the clusterization in form of various graphical representations. Monotone Cluster-Size Graph (MCG) is a visualization that shows clusters based on their cluster size rather than their dependence set size, while Slice/Cluster-Size Graph (SCG) is a combination of the MSG and MCG [12]. Additional views have been introduced in [13], where coloring is applied to represent cluster sizes in the "Heat-map" view and various other code-based views. These approaches, however, do not define a single metric that is alternative to AREA for the investigation of the clusterization.

The international standard on software quality ISO/IEC 9126 provides a classification of quality attributes for software systems and very high level aspects of the measurement of quality [2]. However, these very high level concepts are not directly measurable in existing systems, hence a *model* is required that establishes a connection from the low level measurable metrics like complexity or level of coupling. A common property of these models is that they incorporate the quality aspects of the quality standards, and they provide aggregation mechanisms to gain the higher level quality attributes. The most notable existing models are by Bakota et al. [14], Wagner et al. [15] and by Serebrenik et al. [16]. In this work, we experimented with the probabilistic quality model by Bakota et al., however the other approaches could also be used for this purpose.

B. Software Quality Models

The probabilistic software quality model [14] used in this paper computes the high level quality characteristics based on a directed acyclic graph called the Attribute Dependency Graph – ADG (see Figure 2) – whose nodes correspond to quality properties that can either be internal (low-level) represented by *sensor nodes* in the ADG or external (high-level) represented by *aggregate nodes* in the ADG. Internal quality properties characterize the software product from an internal (developer) view and are usually estimated by using source code metrics. External quality properties give a characterization from an external (end user) view and are usually aggregated in a certain way by using internal and other external quality properties.

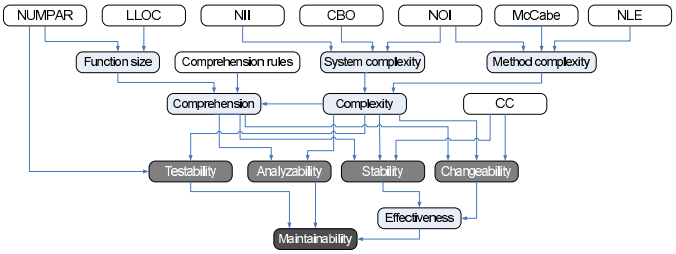

Figure 2. Attribute Dependency Graph (ADG) of the quality model for C.

The aim of the model is to evaluate all the external quality properties by performing an aggregation along the edges of the graph. For each node in the ADG a *goodness value* is calculated from the [0,1] interval (1 is the best). As the basis of the qualification, the probabilistic statistical aggregation algorithm uses a so called benchmark which is a source code metric repository database of approximately 1 MLOC in size.

III. Clusterization vs. Software Quality

A. Correlation and Mutual Information

The most common way to discover a statistical relationship between two random variables is using *(Pearson) correlation* ρ_{XY}, whose value is always in $[-1, +1]$. Despite its ubiquitous usage, it can only measure linear relationship between a pair of variables. Moreover, $\rho_{XY} = 0$, when X and Y are statistically independent, but zero correlation does not imply independence.

Another measure of dependence relationship is the entropy-based *mutual information (MI)*. The entropy of a random variable X is defined as $H(X) = -\sum_{x\in X} p(x)\log_2 p(x)$, where $p(x)$ is a (discrete) marginal probability distribution of X. The mutual information of X and Y is

$$I(X,Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2\left(\frac{p(x,y)}{p(x)p(y)}\right),$$

where $p(x,y)$ is the joint probability distribution of X and Y. Mutual information can be thought of as the reduction of the uncertainty of one variable given knowledge of the other variable. Unlike correlation, it can detect general dependence, not only linear one. It has become a widely used test in bioinformatics, in neuroscience and other areas of sciences and engineering recently [4], [17]. The drawback of mutual information is that usually we do not know the distributions of the experimental data, and the estimation from finite samples may be affected by systematic errors.

B. Clusterization Metrics

There is a large number of possible clusterization metrics that could be defined. For the purpose of the early investigation reported here, the following three were used:

Area under MSG (AREA): Sum of the sizes of dependence sets relative to the maximum possible sum.
The apparent weakness of AREA is that it increases if all dependence sets are increased by the same amount, although intuitively clusterization should not be different in such cases.

Regularity of sets (REGU): This metric tries to alleviate the above effect by counting the number of missing dependence set sizes relative to the number of elements.
The value of REGU is minimal (*i. e.* zero) if all dependence sets are of different size, and maximal if all sets are of the same size.

Entropy-based (ENTR): It is calculated using the entropy formula $H(X)$ with normalization. The probability values are determined by placing the dependence sets into bins based on their sizes and substituting the resulting bin frequencies in $p(x)$.

C. Subject Programs

We collected a set of C programs for analysis that were chosen from different domains and their sizes varied in a wide range as shown in Table I. The last column contains the degree of clusterization (No/Large/Enormous) as determined by visually inspecting the MSGs of the programs, *i. e.* assessing the width of the plateaus formed relative to the program size.

The three clusterization metrics were compared based on how much they agree with our judgement regarding clusterization. It turned out that AREA and ENTR give nearly the same results (except that low ENTR means high AREA and vice versa). This is somewhat surprising as their definitions are quite different. Regarding REGU, its values are quite high on all programs. It agrees well with the other two metrics on the most clustered programs, but could not distinguish the less clustered programs from each other well.

Table I
SUBJECT PROGRAMS WITH DEGREE OF CLUSTERIZATION

Program	Methods	AREA	ENTR	REGU	Clusterization
espresso	788	2.5	95.2	87.8	No
tile-forth	301	4.8	95.8	85.7	No
epwic v1	152	7.2	90.3	79.6	No
findutils v4.2.31	608	7.5	82.6	86.8	No
barcode v0.98	58	8.4	84.7	84.5	Large
byacc v1.9	188	8.5	84.7	79.8	Large
gnugo v3.6	3,020	10.2	89.8	90.8	Large
gnubg v0.9.0	1,589	11.0	89.7	79.4	No
flex v2.4.7	148	12.6	92.6	70.3	No
diffutils v2.7	180	15.2	83.3	73.3	No
gnuchess v5.08	258	18.2	86.9	75.2	Large
compress	24	19.1	82.3	75.0	Large
termutils v2	74	19.9	82.0	64.9	Large
userv v0.95	232	20.9	89.0	69.0	No
a2ps v4.14	1,067	25.2	81.6	82.9	Large
interpreter	105	26.5	82.5	80.0	Large
time v1.7	19	31.3	79.2	73.7	Large
ctags v5	516	33.2	61.8	83.1	Large
wdiff v0.5	39	33.8	74.0	74.4	Large
li	358	38.9	83.2	90.5	Large
sendmail v8.14.0	863	40.8	75.6	82.7	Large
bc v1.06	186	42.5	64.3	75.8	Large
ftpd v1.0.29	247	45.4	47.0	89.1	Large
acct v6.3	48	60.7	66.8	70.8	Large
nascar v0.8.0	23	64.1	59.3	69.6	Large
indent v2.2.9	111	71.1	29.9	88.3	Enormous
ed v0.8	120	83.7	30.5	92.5	Enormous
go	383	91.0	18.0	97.7	Enormous
sudoku v1.11	38	92.5	9.0	94.7	Enormous
copia	242	99.9	0.5	99.6	Enormous

Clusterization metric values shown as percentages, ordered by AREA.

Table II
RELATING THE AREA CLUSTERIZATION AND QUALITY METRICS FOR SOME TYPICAL QUALITY METRICS

AREA vs. QM	Complexity	Function size	NLE	CBO	NOI
Correlation	-0.102	0.251	0.284	0.330	-0.481
MI (5 bins)	0.401	0.338	0.343	0.233	0.276
MI (10 bins)	0.451	0.489	0.470	0.479	0.493

NLE: Nesting Level
CBO: Coupling Between Object classes
NOI: Number of Outgoing Invocations

IV. MEASUREMENTS

Applying the quality model introduced earlier we acquired low- and high-level software quality attributes for the programs in Table I, these were compared with the clusterization metrics using correlation and normalized mutual information $R_{min}(X,Y) = I(X,Y)/\min(H(X),H(Y))$ that yields values in $[0,1]$.

A. Correlation and AREA

Usually the strength of correlation ρ is determined by the following approximate ranges: *(almost) no correlation* for $0 \le |\rho| \le 0.1$; *small* for $0.1 < |\rho| \le 0.3$; *medium* for $0.3 < |\rho| \le 0.5$; and *large* for $0.5 < |\rho| \le 1$.

Correlation showed (see first row of Table II) that AREA and NOI are close to have large correlation, CBO has medium correlation, Function Size and NLE have correlation close

to being medium. The other quality attributes showed small correlation with AREA, less than 0.2 each (only Complexity is shown). Note that only sensor (*i. e.* low-level) attributes have at least medium correlation with AREA.

B. Mutual information (MI) and AREA

As mentioned before, small correlation does not mean that the corresponding variables are statistically independent, only that *linear dependence* between the variables is "small", so we also used mutual information between AREA and all quality properties in order to further explore dependencies. Mutual information $I(X, Y)$ tells us how much we know about Y by knowing X. A small value means that X and Y are close to being statistically independent, not only linearly independent.

We determined the probability distribution functions for entropy calculation by dividing the $[0, 1]$ interval into 5 and 10 equal-length bins, and counting the frequency of data in the bins (maximum likelihood method). Since the dimensionality of the properties is 30, it is clear that even more bins would only give false results. Using 5 bins underestimates, while 10 bins overestimate dependency. For determining the degree of dependency, approximately the same thresholds can be used as the ones for correlation (see above).

Table II contains typical mutual information results for a limited number of quality attributes. We can determine from the full set of results that nearly all of those quality metrics that correlated with AREA show a considerable degree of dependence using normalized mutual information. Also, a number of quality metrics not correlated with AREA exhibit a certain degree of dependence, including Complexity, Maintainability, Analyzability, Stability, Changeability and Comprehension.

Since AREA has large entropy while these properties have smaller entropy, one can conclude that knowing AREA gives a substantial amount of information on any of the above properties. This suggests that there is a considerable dependence between AREA and such properties that were missed by correlation computations. Hence, mutual information could improve our understanding of the relationship of the quality properties and AREA, even though the amount of data is relatively small.

C. Mutual information (MI) and REGU

The mutual information values were also calculated for REGU and the quality attributes. We cannot present these due to space constrains, but the numbers suggest that REGU might be a better predictor for quality than AREA for programs that are heavily clusterized.

V. CONCLUSIONS

Empirical evidence from earlier reports showed that the degree of clusterization in programs is related to various aspects of software quality. This paper presented a first step towards better understanding clusterization by comparing it to quality model-based attributes. Interestingly, **significant correlation could be identified only with low level metrics**, however **mutual information analysis showed a relationship also with some of the higher level attributes**. This result suggests that the latter method could be used in future studies instead of simple correlation, but this also imposes a validity issue of the model itself. This definitely needs to be investigated further, for which significantly more measurement data are required.

Another interesting observation of the study was that on average the AREA metric still seems to be a solid indicator of clusterization, but the finding about the REGU metric for programs with high clusterization requires further investigation. It is conceivable that for particular types of programs above a certain size **REGU might be a better clusterization, and hence, quality indicator than AREA**.

REFERENCES

[1] M. Harman, D. Binkley, K. Gallagher, N. Gold, and J. Krinke, "Dependence clusters in source code," *ACM Transactions on Programming Languages and Systems*, vol. 32, no. 1, pp. 1–33, Nov. 2009.

[2] ISO/IEC, *ISO/IEC 9126. Software Engineering – Product quality*. ISO/IEC, 2001.

[3] D. Binkley and M. Harman, "Locating dependence clusters and dependence pollution," in *Proceedings of the 21st International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 177–186.

[4] R. Steuer, J. Kurths, C. Daub, J. Weise, and J. Selbig, "The mutual information: Detecting and evaluating dependencies between variables," *Bioinformatics*, vol. 18 Suppl.2, pp. S231–S240, 2002.

[5] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 1, pp. 26–61, 1990.

[6] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in *Proceedings of the 33rd ACM SIGSOFT International Conference on Software Engineering (ICSE)*, 2011, pp. 746–765.

[7] L. Schrettner, J. Jász, T. Gergely, Á. Beszédes, and T. Gyimóthy, "Impact analysis in the presence of dependence clusters using Static Execute After in WebKit," in *Proceedings of the 12th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'12)*, Sep. 2012, pp. 24–33.

[8] Á. Hajnal and I. Forgács, "A demand-driven approach to slicing legacy COBOL systems," *Journal of Software: Evolution and Process*, vol. 24, no. 1, pp. 67–82, Jan. 2012.

[9] D. Binkley, N. Gold, M. Harman, Z. Li, K. Mahdavi, and J. Wegener, "Dependence anti patterns," in *4th International ERCIM Workshop on Software Evolution and Evolvability (Evol'08)*, 2008, pp. 25–34.

[10] S. Black, S. Counsell, T. Hall, and D. Bowes, "Fault analysis in OSS based on program slicing metrics," in *Proceedings of the EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, 2009, pp. 3–10.

[11] D. Binkley and M. Harman, "Identifying 'linchpin vertices' that cause large dependence clusters," in *Proceedings of the Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'09)*, 2009, pp. 89–98.

[12] S. S. Islam, J. Krinke, D. Binkley, and M. Harman, "Coherent dependence clusters," in *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (PASTE'10)*, 2010, pp. 53–60.

[13] S. S. Islam, J. Krinke, and D. Binkley, "Dependence cluster visualization," in *Proceedings of the 5th international symposium on Software visualization (SOFTVIS'10)*, 2010, pp. 93–102.

[14] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, "A Probabilistic Software Quality Model," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM 2011)*. Williamsburg, VA, USA: IEEE Computer Society, 2011, pp. 368–377.

[15] S. Wagner, "A bayesian network approach to assess and predict software quality using activity-based quality models," *Information and Software Technology*, vol. 52, no. 11, pp. 1230–1241, 2010.

[16] A. Serebrenik and M. van den Brand, "Theil index for aggregation of software metrics values," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'10)*, 2010, pp. 1–9.

[17] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991.